Ejercicio de Ajuste

A continuación, intentemos ajustar la función `y(x) = ax + b`.

```
import numpy as np
import math
```

x = np.linspace(-math.pi, math.pi, 20)

```
print(x)
```

```
[-3.14159265 -2.81089869 -2.48020473 -2.14951076 -1.8188168  -1.48812284
 -1.15742887 -0.82673491 -0.49604095 -0.16534698  0.16534698  0.49604095
  0.82673491  1.15742887  1.48812284  1.8188168   2.14951076  2.48020473
  2.81089869  3.14159265]
```

Noté que es `linspace`, no `linespace`. Esto es parte del código de un ejemplo en un tutorial de `PyTorch`. Estos números decimales pueden no ser muy intuitivos.

```
x = np.linspace(0, 100, 20)
```

```
import numpy as np
import math

x = np.linspace(0, 100, 20)
y = np.linspace(0, 100, 20)

print(x)
print(y)
```

Así obtenemos dos conjuntos de datos. ¿Cómo podemos representarlos gráficamente?

Sin embargo, resulta que `x` e `y` son iguales.

```
x = np.random.rand(2)
print(x)
```

```
[0.06094295 0.89674607]
```

Sigue haciendo cambios.

```
x = np.random.rand(2)*100
print(x)
```

```
[39.6136151  66.15534011]
```

Continúa con las modificaciones.

```
import numpy as np
import math
import matplotlib.pyplot as plt

x = np.random.rand(10)*100
y = np.random.rand(10)*100

plt.plot(x, y)
plt.show()
```

```
[20.1240488  59.69327146 58.05432614  3.14092909 82.86411091 43.23010476
 88.09796699 94.42222486 58.45253048 51.98479507]
[58.7129098   1.6457994  49.34115933 71.13738592 53.09736099 15.4485691
 45.12200319 20.46080549 67.48555147 91.10864978]
```

Se puede observar que va de `(20.1,58.7)` a `(59.7,1.6)` y luego a `(58,49.3)`. Aunque el gráfico parece desordenado, sigue teniendo un patrón. Es un trazo continuo.

```
import numpy as np
import math
import matplotlib.pyplot as plt

x = np.random.rand(2)*100
y = np.random.rand(2)*100

print(x)
print(y)

plt.plot(x, y)
plt.show()
```

Notamos que las escalas de `x` e `y` siempre están cambiando. Por lo tanto, dos líneas que parecen iguales, en realidad no lo son. Entonces, ¿cómo encontramos los valores de `a` y `b` en la ecuación `y(x) = ax + b`? Supongamos que ahora conocemos dos puntos en esta línea. Observamos que podemos resolverlo usando papel de borrador. Restamos las dos ecuaciones, eliminamos `b` y encontramos `a`. Luego, sustituimos `a` en una de las ecuaciones para encontrar `b`.

Sin embargo, ¿se podría utilizar un método de adivinanza? Utilizando el método de bisección. Vamos a intentarlo.

```
import numpy as np
import math
import matplotlib.pyplot as plt

x = np.random.rand(2)*100
y = np.random.rand(2)*100

a_max = 1000
a_min = -1000
b_max = 1000
b_min = -1000

def cal_d(da, b):
    y0 = x[0] * da + b
    y1 = x[1] * da + b
    d = abs(y0 - y[0]) + abs(y1 - y[1])
    return d

def cal_db(a, db):
    y0 = x[0] * a + db
    y1 = x[1] * a + db
    d = abs(y0 - y[0]) + abs(y1 - y[1])
    return d

def avg_a():
    return (a_max + a_min) / 2

def avg_b():
    return (b_max + b_min) / 2
```

```
for i in range(100):
    a = avg_a()
    b = avg_b()
    max_d = cal_d(a_max, b)
    min_d = cal_d(a_min, b)
    if max_d < min_d:
        a_min = a
    else:
        a_max = a

    a = avg_a()
    max_db = cal_db(a, b_max)
    min_db = cal_db(a, b_min)
    if max_db < min_db:
        b_min = b
    else:
        b_max = b

print(x)
print(y)
print('a = ', avg_a())
print('b = ', avg_b())
print(avg_a() * x[0] + avg_b())
print(avg_a() * x[1] + avg_b())
```

Ejecútalo.

```
[42.78912791 98.69284173]
[68.95535212 80.89946202]
a =  11.71875
b =  -953.125
-451.68990725289063
203.4317390671779
```

Sin embargo, los resultados mostraron una gran discrepancia.

Vamos a simplificar el problema. Supongamos que `y(x) = ax`. Dado un conjunto de valores `x`, `y`, queremos encontrar `a`. Aunque podríamos calcularlo directamente, vamos a intentar adivinarlo.

```
import numpy as np
import math
import matplotlib.pyplot as plt
from numpy.random import rand, randint
```

x = randint(100) y = randint(100)

```
a_max = 1000
a_min = -1000

def cal_d(da):
    y0 = x * da
    return abs(y0 - y)

def avg_a():
    return (a_max + a_min) / 2

for i in range(1000):
    a = avg_a()
    max_d = cal_d(a_max)
    min_d = cal_d(a_min)
    if max_d < min_d:
        a_min = a
    else:
        a_max = a

print(x)
print(y)
print(avg_a())
print(avg_a()*x)
```

El resultado es alentador. Adiviné con bastante precisión.

```
96
61
0.6354166666666667
61.00000000000001
```

Sin embargo, normalmente se escribe `for i in range(15):`, lo que significa iterar 15 veces, lo cual es bastante preciso. ¿Por qué? Observa que nuestros valores de `x` e `y` están entre 0 y 100. Por lo tanto, el valor de `a` también está entre 0 y 100. Por ejemplo, `x=1, y=99` y `x=99, y=1`. Así que los valores iniciales de `a_min` y `a_max` se pueden optimizar. Nota que `1/99` es `0.01`. Por lo tanto, para alcanzar una precisión de `0.01`, necesitamos calcular 2^n ≈ 10000. log2(10000) = 13.28. Esto significa que establecerlo alrededor de 14 debería ser suficiente.